UCRL-84102
PREPRINT

GEARS: A Package for the Solution of Sparse, Stiff Ordinary Differential Equations

Andrew H. Sherman
Department of Computer Science
University of Toronto
Toronto, Ontario

Alan C. Hindmarsh
Mathematics and Statistics Section
Lawrence Livermore Laboratory
Livermore, CA 94550

This paper was prepared for presentation at the SIAM International Conference on Electric Power Problems: The Mathematical Challenge, March 18-20, 1980, Seattle.

March 1980

GEARS: A Package for the Solution of Sparse, Stiff Ordinary Differential Equations

Andrew H. Sherman[1] and Alan C. Hindmarsh[2]

A B S T R A C T

This paper describes GEARS, a package of Fortran subroutines designed to solve stiff systems of ordinary differential equations of the form $dy/dt = f(y,t)$ where the Jacobian matrices $J = \partial f/\partial y$ are large and sparse. The integrator is based on the stiffly stable methods due to Gear, and this leads to a sparse system of nonlinear equations at each time step. These are solved with a modified Newton iteration, using one of two separate sparse matrix packages to solve the sparse linear equations that arise. In this paper we describe the package in some detail, discuss a number of issues that affected the design of the package, and present a numerical example to illustrate the effectiveness of the package.

"Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore Laboratory under contract number W-7405-ENG-48."

[1]Department of Computer Science, University of Toronto, Toronto, Ontario, M5S 1A7, Canada.

[2]Mathematics and Statistics Section, Lawrence Livermore Laboratory, Livermore, CA 94550.

GEARS: A Package for the Solution of Sparse Stiff Ordinary Differential Equations

1. Introduction

The mathematical modeling of electric power problems (among others) often leads to initial value problems for systems of ordinary differential equations (ODE's) in time, which can be put in the abstract form

$$\dot{y} \equiv dy/dt = f(y,t), \quad y(t_0) = y_0 \text{ (given)}. \tag{1}$$

Here y and f are vectors of length N, and N may be quite large (thousands) for realistic applications. Often these ODE problems are stiff, and as a result involve the N×N Jacobian matrix

$$J \equiv J(y,t) \equiv \partial f/\partial y \tag{2}$$

in their solutions. Stiffness occurs when one or more of the eigenvalues λ of J are such that the associated time constant $\tau \equiv -1/\mathrm{Re}(\lambda)$ is positive and very small compared to the time span of interest. (Since the λ's can vary with y and t in general, one has to qualify this by stipulating that, for a given particular solution, some τ is small for a significant part of the total time interval beyond any rapid transients that occur in the solution.) Failure to take stiffness into account can result in numerical methods that require time steps on the order of this smallest τ, while the time scale of the solution curves is much longer.

One Fortran software package that has become widely used for both stiff and nonstiff problems generally is the GEAR package [1]. When used for stiff problems of the form (1), it uses a variable-order method (Gear's method) based on the backward differentiation formulas (BDF's),

$$y_n = \sum_{i=1}^{q} \alpha_i y_{n-i} + h\beta_0 \dot{y}_n, \tag{3}$$

of order q, $1 \le q \le 5$. Here y_n is the computed approximation to $y(t_n)$, $\dot{y}_n$ denotes $f(y_n,t_n)$, h is a step size ($h = t_k - t_{k-1}$), and the α_i's and β_0 are constants depending only on q. While this formula assumes a fixed h, GEAR varies both q and h, with changes in h handled by interpolating on the given t mesh to get data on the new t mesh.

Since $\beta_0 \neq 0$, formula (3) is implicit, and each step requires the solution of an algebraic system (generally nonlinear) of the form

$$y_n = h\beta_0 f(y_n, t_n) + a_n. \tag{4}$$

Stiffness is manifested in large elements of $\partial f/\partial y$, and this rules out the classical fixed point or functional iteration on (4) as an efficient means of solution. Instead, a modified Newton iteration is used. This takes the form

$$P\Delta y^{(m)} \equiv P[y_n^{(m+1)} - y_n^{(m)}] = a_n + h\beta_0 f(y_n^{(m)}, t_n) - y_n^{(m)}, \tag{5}$$

where P is a matrix satisfying

$$P \approx I - h\beta_0 J \quad (I = N\times N \text{ identity}). \tag{6}$$

The initial guess $y_n^{(0)}$ (prediction) is obtained from an explicit formula analogous to (3), and the corrector iterates $y_n^{(m)}$ are computed until an appropriate convergence criterion is met.

When implementing (5) and (6), GEAR assumes that J is a full (dense) N×N matrix. It is this feature that limits GEAR to small or modest-sized problems and has motivated the development of variant packages capable of handling larger problems with reasonable efficiency. GEARB [2] is one such variant that is intended for the case in which J is large and banded. The program GEARS described in this paper is a more recent variant that allows for an arbitrary sparseness structure in J.

The importance of the variants of GEAR is due to the fact that many applications lead naturally to systems in which the Jacobian matrix is quite sparse: typically the i-th component $\dot{y}^i$ of $\dot{y} = f(y,t)$ depends on relatively few components y^j of y. This occurs, for example, in network simulation and in time-dependent partial differential equations treated by the method of lines (i.e., finite differencing in space, giving ODE's in time t). For such problems, N may be so large that storing or computing with the full N×N matrix J (or P) (or even its band) is out of the question, even though the number of nonzero elements of J (and hence P) may be quite manageable. This situation calls for the use of general sparse matrix techniques, in conjunction with the general time integration technique represented by GEAR. The GEARS package is the result of such a combination, using the Yale Sparse Matrix Package (YSMP) [3,4].

By way of historical background, a version of GEARS has existed since 1975 [5]. This made use of the version of YSMP that existed at that time [6]. However, further development of YSMP since then made it necessary to heavily revise GEARS. Moreover, many features and options have been included in the present version to make its use as simple as possible for a typical user with only minimal background in sparse matrix techniques. Some of these are discussed later in this paper.

2. Overview of GEAR/GEARS

Both GEAR and GEARS (and other related ODE solvers) share a modular structure which makes the development of variants fairly straightforward. In its simplest form, this structure is shown in Figure 1. The driver handles user communication and the time-stepping process, while the single-step routine handles the taking of a step, including step and order selection, and other aspects which are independent of the assumptions about the structure of J. Matrix-related matters are handled by separate routines called by the above routines, as shown. To obtain GEARS, these latter routines were replaced in GEAR by routines that take account of Jacobian structure, with relatively little change to the other routines in GEAR.

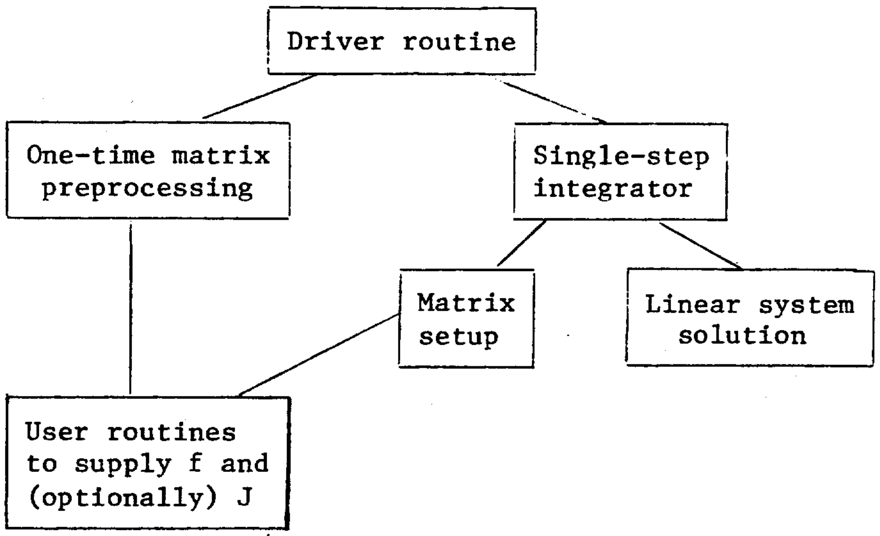
Driver routine
One-time matrix preprocessing
Single-step integrator
Matrix setup
Linear system solution
User routines to supply f and (optionally) J

Figure 1. Modular structure of GEAR and GEARS

From the user's point of view, GEARS resembles GEAR except for a few parts of the user interface associated with sparse matrices. For both packages, the user must supply a subroutine DIFFUN that computes the function $f(y,t)$ in (1). As an option, he may also supply a routine that computes elements of the Jacobian $J = \partial f/\partial y$ in closed form, but if this is not feasible, the package will compute J by difference quotients. The user then writes a calling program which calls the driver and supplies the initial conditions, an error tolerance, a method flag, and a value of t at which answers are desired. Repeated calls with new specified t values are made as desired. The method flag includes a choice between stiff and nonstiff formulas (Adams formulas are used for nonstiff problems), and it tells whether or not J is being supplied. For GEARS, it also indicates whether structure information about the Jacobian will be supplied by the user or computed by GEARS. The only other difference between the user interfaces of GEAR and GEARS is that the latter includes in the call sequence a work array (actually an integer and real copy of the same work space) which is to be devoted to sparse matrix computations. Determining the required length of this array may require some experimentation.

3. Software Aspects of Sparse ODE's

As noted in the Introduction, programs such as GEAR have been quite successful in solving stiff ODE's that are small and dense or moderately large and banded. For certain applications, however, it is essential to more fully exploit the sparseness of the system of ODE's in order to reduce storage or computation time. This section examines several of the key techniques in GEARS that would probably be useful in any package that does this.

There were two basic objectives in designing GEARS: first, retention of the properties of GEAR as an ODE solver while enhancing its efficiency for sparse systems, and second, provision of a user interface allowing for various degrees of user knowledge about sparse matrix techniques. The first goal is necessary in order to allow the simple substitution of GEARS for GEAR. It also facilitates comparisons that reflect on the utility of exploiting sparseness for any particular problem. The second goal is desirable in order to cater to the needs of the large variety of anticipated users, ranging from numerical analysts expert in sparse matrix handling to scientists and engineers wishing to treat GEARS as a "black box".

Two of the primary techniques in GEARS that lead to enhanced efficiency are the use of a special method to compute finite difference approximations to J (when needed) and the use of the Yale Sparse Matrix Package (YSMP) to solve the linear systems in (5). The method for computing the finite difference approximations is due to Curtis, Powell, and Reid [7]. It consists in grouping the components of y into NGP groups such that all the components in a group can be varied simultaneously in computing a difference quotient. Thus only NGP evaluations of f are required to approximate J, instead of the N evaluations used by GEAR. (Usually NGP << N.)

The linear systems in (5) are sparse and usually nonsymmetric. In general, it should be anticipated that partial pivoting will be required when Gaussian elimination is used to factor P into the product LU. However, experience has shown that pivoting is only rarely required in certain applications [8], and, in any case, if the factorization process should fail, a change in the stepsize h will usually improve matters because of the form of P. Since substantial cost is associated with the incorporation of partial pivoting into a subroutine for sparse Gaussian elimination, we have chosen to initially include in GEARS only a package (YSMP) that does no pivoting for numerical stability.

YSMP [3,4] is a set of Fortran subroutines designed to solve sparse linear systems by Gaussian elimination. The package has received extensive use and appears to be quite efficient. The reader is referred to [9] for a discussion (for symmetric systems) of the relative merits of sparse codes such as YSMP and band codes such as that included in GEARB, but we note that there appear to be many situations in which the sparse code is substantially more efficient.

GEARS uses two modules from YSMP to solve the systems (5): ODRV (slightly modified) to select a permutation matrix Q such that QPQ^T can be efficiently factored, and CDRV to actually factor QPQ^T and solve for $\Delta y^{(m)}$. If the matrix QPQ^T has no factorization for the chosen Q (a possibility since ODRV ignores stability in choosing Q), GEARS reduces h (to make P nearer to the identity matrix) and retries the step. In our experience, this has rarely happened.

We now turn to the user interface for GEARS. Among other things, the user of a code based on GEAR may choose to provide the true Jacobian matrix J by including a subroutine to compute it. In the case of GEAR or GEARB it is not difficult for a typical user to compute J and store it in an appropriate manner. In the case of GEARS, however, many users may find it inconvenient to compute J in the form that is most convenient for internal use. As a result, GEARS does not require the user to compute all of J at once. Instead, the user provides a subroutine (PDS) that computes a single column of J and returns it as a vector of length N. (Only the nonzero components of the column need be computed and stored.) GEARS uses structure information about the locations of nonzeroes in J to store the output of PDS in an appropriate form. In addition to simplicity for the user, this scheme has the advantage of being completely independent of the internal data structure for J.

The structure information just mentioned is required whether the user provides PDS or depends on GEARS to compute J with difference quotients. It can be input by the user in two integer arrays IA and JA, or alternatively, it can be computed internally by GEARS (using several evaluations of f or calls to PDS). The former choice is certainly more efficient, but it requires more expertise on the part of the user. The latter choice is expensive, but it is usually done only once per problem, so its simplicity may outweigh its cost. Whichever option is chosen initially, the user may later signal GEARS (during the integration) either to accept new arrays IA and JA or to recompute the structure information.

4. An Example Problem

To illustrate the potential benefits of GEARS over other members of the GEAR family of codes, consider the parabolic differential equation

$$u_t = \nabla\cdot(D(e^u)\nabla u) + D(e^u)(u_x^2+u_y^2). \tag{7}$$

This equation arises after the variable transformation $u = \ell n\,\alpha$ from the equation

$$\alpha_t = \nabla\cdot(D(\alpha)\nabla\alpha) \tag{8}$$

that can be used to model the impurity atom distribution near the edge of a diffusion mask in the fabrication process for narrow-channel MOS transistors [10]. In both equations, D(z) is defined by

$$D(z) = D_0\left[\frac{1+\beta(z+\sqrt{z^2+1})}{1+\beta}\right]\left[1+\frac{z}{\sqrt{z^2+1}}\right] \tag{9}$$

where D_0 and β are physical constants. For the problem studied here, $D_0 = 1$ and $\beta = 100$.

The equation (7) was solved on the rectangular domain [-17,0]x[-17,17]. Appropriate boundary conditions for a constant-surface-concentration model are

$$\left.\begin{array}{lll}
u = \ell n\ 20 & x = 0, & 0 \le y \le 17 \\
u = \ell n(2\mathrm{x}10^{-4}) & x = -17, & -17 \le y \le 17 \\
u = \ell n(2\mathrm{x}10^{-4}) & -17 \le x \le 0, & y = -17 \\
u_x = 0 & x = 0, & -17 \le y \le 0 \\
u_y = 0 & -17 \le x \le 0, & y = 17
\end{array}\right\} \tag{10}$$

The initial condition chosen had $u = \ell n(2\mathrm{x}10^{-4})$ on the interior of the domain and satisfied the boundary conditions (10). The integration was carried out between times t=0 and t=1.

To solve (7) numerically, the method of lines was used on a regular rectangular spatial mesh with equal spacing in the x and y directions. The righthand side of (7) was discretized using central differences for the self-adjoint term and upstream differences for u_x^2 and u_y^2. The resulting discretization is first order accurate and yields the system of ODE's

$$dy/dt = f(y,t), \tag{11}$$

where the components of y correspond to approximations to u at the mesh nodes, and an evaluation of f is simply an evaluation of the difference approximation to the righthand side of (7) (subject to (10)) for a given set of nodal values.

The discrete problem was solved for mesh spacings of 17/10 and 17/19, giving systems of 189 and 702 ODE's, respectively. The corresponding Jacobian matrices have half-bandwidths of 10 and 19, respectively. Both GEARB and GEARS were used with an error tolerance of 10^{-5} and an initial step size of 10^{-6}. Both codes obtained Jacobian matrices using difference quotients. The computed solutions of the system (11) were identical.

The results are reported in Table 1. In the experiments, GEARS was run both with structure information supplied (denoted by GEARS) and with structure information computed using N+1 function evaluations (denoted by GEARS*). The statistics given in the table are the number of integration steps (NST), number of function evaluations (NFE), number of Jacobian evaluations (NJE), number of function evaluations per Jacobian evaluation (NGP), number of function evaluations used in Jacobian or structure evaluation (NFJ), time in seconds on a CDC7600 using the CHAT compiler (TIME), and the number of words of method-dependent storage (MDS). (In addition, a certain amount of storage is required by both GEARB and GEARS. For N = 702, this is approximately 8,000 words.)

TABLE 1

N	CODE	NST	NFE	NJE	NGP	NFJ	TIME	MDS
189	GEARB	101	386	10	21	210	6.04	5,859
	GEARS	101	246	10	7	70	4.64	7,935
	GEARS*	101	436	10	7	172	6.75	7,935
702	GEARB	193	938	15	39	585	59.85	40,716
	GEARS	193	458	15	7	105	36.97	35,265
	GEARS*	193	1,161	15	7	808	66.04	35,265

The results shown in Table 1 indicate that GEARS can be substantially more efficient that GEARB. However, two comments are required to place the results in perspective. First, most of the time differences are due to the number of function evaluations required, although for these problems, GEARS seems to be 10-15% faster than GEARB excluding function evaluations. (This problem is well-suited to GEARB since a rectangular mesh is used.) Second, the problem is not very stiff for the mesh spacings used; a finer spatial mesh or a longer time interval would yield a stiffer system of ODE's and enhance the advantage of GEARS. (The cost of structure computation in GEARS would also become relatively less important.)

5. Extensions

GEARS should be viewed not as the end of a line of research, but only as a substantial first step. Numerous issues remain to be investigated, and substantial changes (hopefully improvements) are certain. To indicate the direction of the work, some of these are listed here.

i) The question of partial pivoting must be examined in more detail. For those applications requiring pivoting, GEARS could be extended to include an appropriate sparse matrix package and a means of switching between it and YSMP.

ii) Currently, GEARS re-evaluates J whenever P is re-evaluated, even if only the order or step-size has changed. While costly, this saves the storage that would be required to save both P and J internally. Recently, we have experimented with techniques that recover J from P, and thereby avoid the extra evaluations of J. We expect to incorporate such a technique into GEARS in the near future.

iii) GEARS only solves explicit systems of ODE's. Implicit systems of the form

$$A\dot{y} = f(y,t)$$

arise naturally in certain applications of the method of lines and elsewhere. GEARS could be extended to handle such systems.

iv) There are several drawbacks (from an ODE point of view) to the current GEARS interface. A new interface could be designed to provide more flexibility to the user in terms of error control and other aspects of the ODE solution process.

v) Several recently proposed sparse matrix techniques could be adapted for GEARS to allow the sparseness of the system of ODE's to be more fully exploited. One idea suggested by Carver [11] involves ignoring small components of J. This or some similar idea may be useful in GEARS.

References

[1] A.C. Hindmarsh, GEAR: Ordinary Differential Equation System Solver, Lawrence Livermore Laboratory Report UCID-30001, Rev. 3, December 1974.

[2] A.C. Hindmarsh, GEARB: Solution of Ordinary Differential Equations Having Banded Jacobian, LLL Report UCID-30059, Rev. 2, June 1977.

[3] S.C. Eisenstat, M.C. Gursky, M.H. Schultz, and A.H. Sherman, Yale Sparse Matrix Package: I. The Symmetric Codes, Research Report No. 112, Dept. of Computer Sciences, Yale University, 1977.

[4] S.C. Eisenstat, M.C. Gursky, M.H. Schultz, and A.H. Sherman, Yale Sparse Matrix Package: II. The Nonsymmetric Codes, Research Report No. 114, Dept. of Computer Sciences, Yale University, 1977.

[5] J.W. Spellmann and A.C. Hindmarsh, GEARS: Solution of Ordinary Differential Equations Having a Sparse Jacobian Matrix, LLL Report UCID-30116, August 1975.

[6] A.H. Sherman, Yale Sparse Matrix Package User's Guide, LLL Report UCID-30114, August 1975.

[7] A.R. Curtis, M.J.D. Powell, and J.K. Reid, On the Estimation of Sparse Jacobian Matrices, J. Inst. Math. Applic. 13, (1974), pp. 117-119.

[8] A.R. Curtis. Private communication.

[9] D.J. Rose, A.H. Sherman, R.E. Tarjan, and G.F. Whitten, Algorithms and Software for In-Core Factorizations of Sparse, Symmetric, Positive-Definite Matrices, Computers and Structures (1980), to appear.

[10] D.D. Warner and C.L. Wilson, Two-Dimensional Concentration Dependent Diffusion, Bell System Tech. J., 59 (1980), pp. 1-41.

[11] M. Carver. Private communication.